UNIVERSITY OF ASIA PACIFIC

Department of Computer Science and Engineering

Course Title :

Artificial Intelligence and Expert Systems Lab

Course Code : CSE 404

Project : 01

Submitted By:

Shawon Barman

Section: A

ID: 18201043

Submitted to:

Dr. Nasima Begum

Assistant Professor

Department of CSE, University Of Asia Pacific

Project Name: Implementation of a small address map using A* search algorithm

Introduction:

The project problem is to implementation of a small address map from my home to UAP, using A* search algorithm and then find out the optimal path. A* algorithm is a searching algorithm that searches for the shortest path between the initial state to the final state. In this project, I will find the most optimal path from my home (Brahmonkitta road) to my university (UAP) using A* search algorithm.

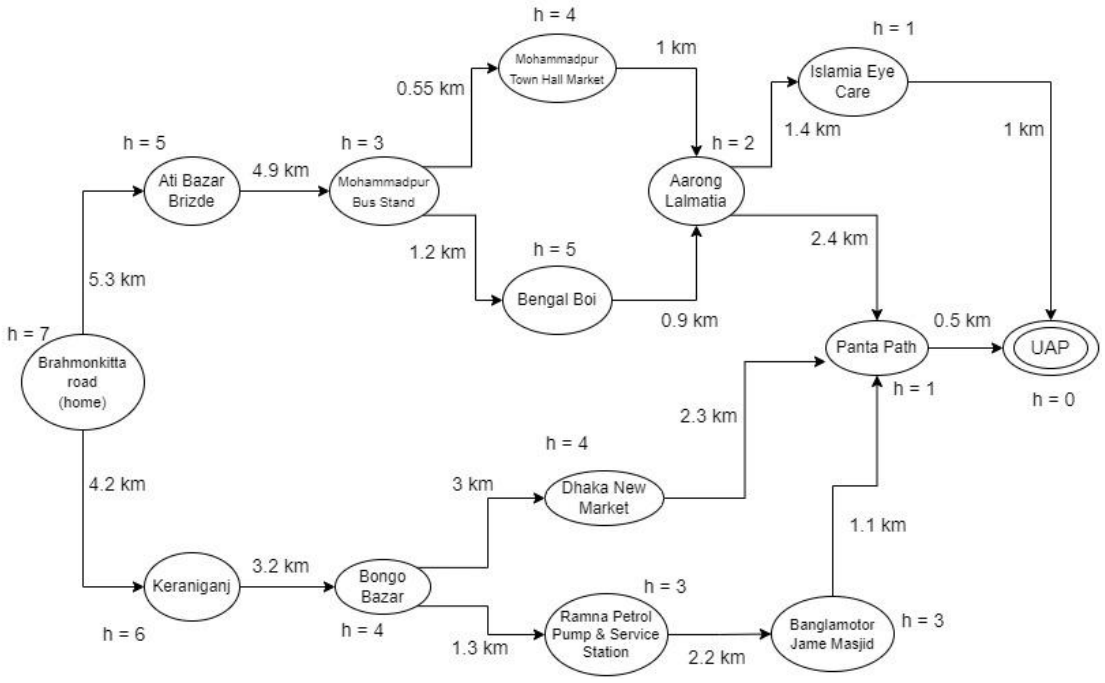
Objective:

The objective of this project is to find an optimal path from my home (Brahmonkitta road) to my university (UAP).

Tools And Languages:

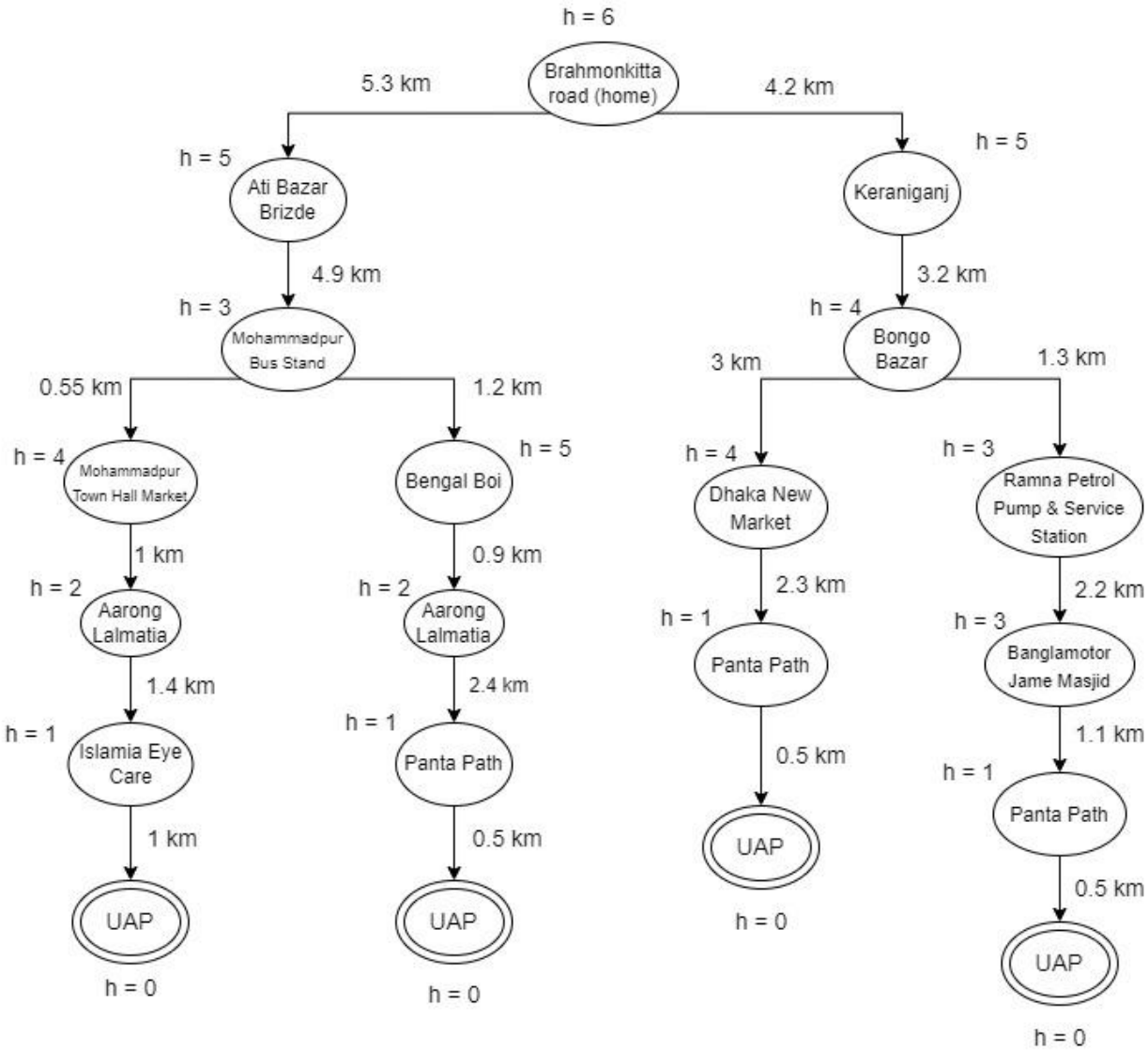
- **Distance Measurement:** Google Maps
- **Map Designing:** Draw.io
- **IDE:** PyCharm
- **Programing Language:** Python

Designed Map:



Here,
Start Node : Shawon’s house
Goal Node : UAP
Cost in Distance : kilometer (km)

Search tree of designed map:



Implemented using python:

```python
def a_star_search(start, goal):
        open_fringe = set(start)
        close_fringe = set()
        g = {} #store distance from starting node
        parents = {}# parents contains an adjacency map of all nodes

        #ditance of starting node from itself is zero
        g[start] = 0

        #start is root node i.e it has no parent nodes
        #so start is set to its own parent node
        parents[start] = start   #start node

        while len(open_fringe) > 0:
            n = None
            #node with lowest f() is found
            for v in open_fringe:
                if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                    n = v

```

```python
            if n == goal or Graph_nodes[n] == None:
                pass
            else:
                for (m, weight) in get_neighbors(n):
                    #nodes 'm' not in first and last set are added to first
                    #n is set its parent
                    if m not in open_fringe and m not in close_fringe:
                        open_fringe.add(m)
                        parents[m] = n
                        g[m] = g[n] + weight


                    #for each node m,compare its distance from start i.e g(m) to the
                    #from start through n node
                    else:
                        if g[m] > g[n] + weight:
                            #update g(m)
```

```python
                                g[m] = g[n] + weight
                                #change parent of m to n
                                parents[m] = n

                                #if m in closed set,remove and add to open
                                if m in close_fringe:
                                    close_fringe.remove(m)
                                    open_fringe.add(m)

            if n == None:
                print('Path does not exist!')
                return None

            # if the current node is the goal
            # then  begin reconstructing the path from it to the start
            if n == goal:
                path = []
                path_cp = []
                full = {
```

```
                'H': "Brahmonkitta road (Home)",
                'ABB': "Ati Bazar Brizde",
                'MBS': "Mohammadpur Bus Stand",
                'MTHM': "Mohammadpur Town Hall Market",
                'BB': "Bengal Boi",
                'AL': "Aarong Lalmatia",
                'IEC': "Islamia Eye Care",
                'PP': "Panta Path",
                'K': "Keraniganj",
                'BBr': "Bongo Bazar",
                'DNM': "Dhaka New Market",
                'RPPSS': "Ramna Petrol Pump & Service Station",
                'BJM': "Banglamotor Jame Masjid",
                'U': "UAP"
                }
                while parents[n] != n:
                    path.append(n)
                    path_cp.append(full[n])
                    n = parents[n]
```

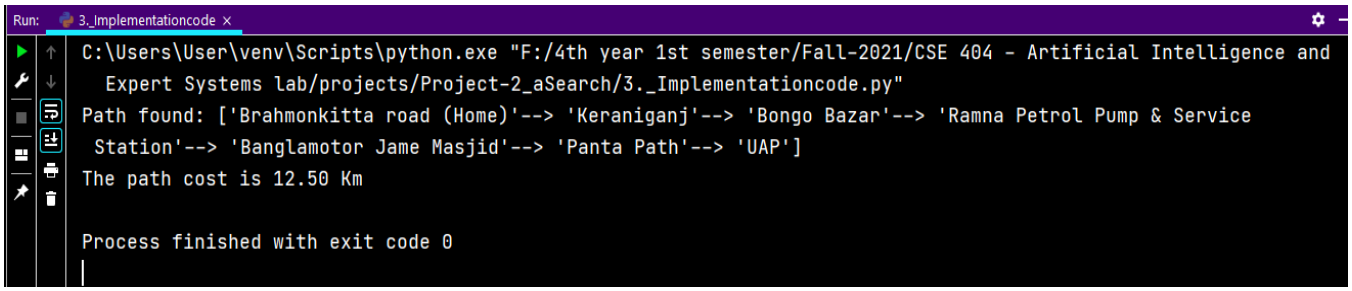
```

                path.append(start)
                path_cp.append(full[start])
                path.reverse()
                path_cp.reverse()
                print('Path found: {}'.format(str(path_cp).replace(",","-->")))
                return path

            open_fringe.remove(n)
            close_fringe.add(n)

        print('Path does not exist!')
        return None

def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
```

```

def heuristic(n):

        H_dist = {
            'H': 7,
            'ABB': 6,
            'MBS': 3,
            'MTHM': 4,
            'BB': 5,
            'AL': 2,
            'IEC': 1,
            'PP': 1,
            'K': 6,
            'BBr': 4,
            'DNM': 4,
            'RPPSS': 3,
            'BJM': 3,
            'U': 0
```

```
        }
        return H_dist[n]

Graph_nodes = {
        'H': [('ABB', 5.3), ('K', 4.2)],
        'ABB': [('MBS', 4.9)],
        'MBS': [('MTHM', 0.55), ('BB', 1.2)],
        'MTHM': [('AL', 1)],
        'BB': [('AL', 0.9)],
        'AL': [('IEC', 1.4), ('PP', 2.4)],
        'K': [('BBr', 3.2)],
        'BBr': [('DNM', 3), ('RPPSS', 1.3)],
        'DNM': [('PP', 2.3)],
        'RPPSS': [('BJM', 2.2)],
        'BJM': [('PP', 1.1)],
        'IEC': [('U', 1)],
        'PP': [('U', 0.5)],
        'U': None
}
```

```
path = a_star_search('H', 'U')

path_cost = 0.0

for i in range(len(path)-1):
    for key, value in Graph_nodes[path[i]]:
        if key == path[i+1]:
            path_cost += value
            break
print("The path cost is %.2f Km" % path_cost)
```

Output:



Result Analysis:

After Using A* Search Algorithm on this designed map, on output we can find the shortest path:

Brahmonkitta road (home) -> Keraniganj -> Bongo Bazar -> Ramna Petrol Pump & Service Station -> Banglamotor Jame Masjid -> Panta Path -> UAP

So, we can say that that is the most optimal and shortest path.

Conclusion:

In this project, after successful implementation, A* search algorithm gives the most optimal path as output. In conclusion, A* search algorithm is a powerful and beneficial algorithm with all the potential. So we can use this algorithm for approximate the shortest path in real-life situation, like - in maps, games etc.